A Comparison Between Break-Even Volatility and Deep Hedging For Option Pricing

Quintin Claassen

A dissertation submitted to the Faculty of Commerce, University of Cape Town, in partial fulfilment of the requirements for the degree of Master of Philosophy.

September 11, 2022

MPhil in Mathematical Finance,
University of Cape Town.

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the Degree of Master of Philosophy in the University of the Cape Town. It has not been submitted before for any degree or examination in any other University.

September 11, 2022

Abstract

The Black-Scholes (1973) closed-form option pricing approach is underpinned by numerous well-known assumptions (see (Taleb, 1997, pg.110-111) or (Wilmott, 1998, ch.19)), where much attention has been paid in particular to the assumption of *constant volatility*, which does not hold in practice (Yalincak, 2012). The standard in industry is to use various volatility estimation and parameterisation techniques when pricing to more closely recover the market-implied volatility skew. One such technique is to use Break-Even Volatility (BEV), the method of retrospectively solving for the volatility which sets the hedging profit and loss at option maturity to zero (conditional on a single, or set of, stock price paths).

However, using BEV still means pricing using existing model frameworks (and using the assumptions which come with them). The new paradigm of Deep Hedging (DH) (as explored by Buehler *et al.* (2019)), ie. using deep neural networks to solve for optimal option prices (and the respective parameters needed to hedge these options at discrete time steps), has allowed the market-maker to go 'model-free', in the sense of being able to price without any prior assumptions about stock price dynamics (which are needed in the traditional closed-form pricing approach).

Using simulated stock price data of various model dynamics, we first investigate whether DH is more successful than BEV in recovering the model implied volatility surface. We find both to perform reasonably well for time-homogeneous models, but DH struggles to recover correct results for time in-homogeneous models. Thereafter, we analyse the impact of incorporating risk-aversion for both approaches only for time-homogeneous models. We find both methods to produce pricing results inline with varying risk aversion levels. We note the simple architecture of our DHNN as a potential point of departure for more complex neural networks.

Acknowledgements

With thanks to my supervisor, Obeid Mahomed, for the guidance along the way; Selwyn Steyn for illustrating Figure 3.1; and every person who asked me when this dissertation would be done.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of pricing and hedging options is one which has challenged market makers since they were first written (Hafner and Zimmermann, 2009, pg.471-486), (Merton, 1973). The first major rigorous inroads were made by the seminal Black-Scholes (1973) option-pricing framework. However, the Black-Scholes closed-form approach is underpinned by numerous well-known assumptions (see (Taleb, 1997, pg.110-111) or (Wilmott, 1998, ch.19)). Much attention has been paid in particular to the assumption of *constant volatility* (Yalincak, 2012). We can calculate the market-implied volatility skew (see Wilmott *et al.*, 1995, pg.52-54) and see that volatility is not constant across option strikes nor across option tenors. Standard industry practice is to use various volatility estimation and parameterisation techniques when pricing to more closely recover the market-implied volatility skew. One such technique common in industry is to use Break-Even Volatility (BEV), the method of retrospectively solving for the volatility which sets the hedging profit and loss at option maturity to zero (conditional on a single, or set of, stock price paths).

However, using BEV still means pricing using existing model frameworks (and using the assumptions which come with them). The new paradigm of Deep Hedging (DH) (as explored by Buehler *et al.* (2019)), ie. using deep neural networks to solve for optimal option prices (and the respective parameters needed to hedge these options at discrete time steps), has allowed the market-maker to go 'model-free', in the sense of being able to price without any prior assumptions about stock price dynamics (which are needed in the traditional closed-form pricing approach).

First we will examine the literature regarding the way volatility used in option pricing is handled in practice, and how it has evolved to its present day treatment (with a particular focus on BEV). Thereafter, we will explore the literature regarding the new DH approach, which uses machine learning techniques in its 'model-free' pricing. An outline of the structure of the remainder of the dissertation then follows.

1.1 Literature Review

1.1.1 The Evolution of Volatility

Volatility is "*the* single most important determinant of an option's value that we do not accurately know" (Wilmott, 2013, pg.756). The treatment of volatility in the pricing process has evolved roughly as follows:

1. Using constant σ or deterministic volatility $\sigma(t)$ (as a function of time) under the Black-Scholes-Merton (1973),(1973) framework.

2. Local volatility as described by Dupire (1994) where volatility $\sigma(t, S_t)$ (a function of time and the stock price at that time) is still deterministic since there is no extra source of randomness. Local volatility fits a unique volatility surface whilst preserving no-arbitrage.

3. Stochastic volatility $\sigma(t, \omega)$ where $\omega \in \Omega$, i.e. a separate process from the underlying (and thus a new source of risk). Such models include the Heston (1993), SABR (2002) and log-volatility (as an Ornenstein-Uhlenbeck process) (2000).

4. Stochastic volatility with jumps, which will mean that stock price paths are no longer continuous. Bakshi *et al.* (1997)'s model and the SVJ-Jump extension of Matytsin (1999) were developed to account for short-end skews being inaccurately described by stochastic volatility (Matytsin, 2000).

5. Rough Volatility as explored by Bayer *et al.* (2016) and Livieri *et al.* (2018), for example, where log-volatility is assumed to follow a fractional Brownian motion with Hurst parameter smaller than 0.5.

It is well known that we can generate a volatility skew (an empirical implied-volatility surface as a function of option strike and time to expiry) based on current market quotes (see Wilmott, 1998, pg.287-288). Particularly, the Stochastic Volatility Inspired (SVI) approach developed by Gatheral and Jacquier (2011) gives a remarkably smooth parametrization of the surface which falls neatly within observed bid-offer spreads. However, there is a divergence between market-implied and model-calibrated surfaces (see Gatheral, 2011, pg.41). Bayer *et al.* (2016) find their Rough Fractional Stochastic Volatility model to be "remarkably consistent" with real-world data, and Livieri *et al.* (2018) go further, stating that volatility *is* rough, according to their findings also using real-world financial time series data.

1.1.2 Break-Even Volatility

The challenges posed by parameterizations of the volatility skew have led to market practitioners exploring the non-parametric method of BEV (Mitoulis, 2019). Crawford (2019) presents a thorough analysis of BEV and the data-sparsity and computational difficulties one can face when performing the calculation, concluding that BEV is a viable method to use for models which are time-homogeneous. However, the results for time-inhomogeneous models were less clear.

Assume that we delta-hedge an option written on an underlying stock according to a chosen model framework: this will result in hedging-error which will either be a profit or loss at maturity. Instead of solving for the volatility in a chosen model which sets the option price equal to observed market quotes, BEV is the volatility obtained by retrospectively solving for that volatility which sets the hedging profit and loss at maturity to zero (conditional on a single, or set of, stock price paths). BEV may be interpreted as the fair volatility to both sides of the option contract (under the Black-Scholes-Merton model). If BEV is calculated across all terms and strikes, it will yield a volatility surface.

This approach was formalised by the work of Dupire (2006). Assume that we have written an option $C(t, S_t)$ on an underlying asset S_t (driven by a source of randomness W_t) with dynamics:

$$dS_t = \mu_t S_t dt + \sigma_t S_t dW_t \tag{1.1}$$

where $\mu_t, \sigma_t \geq 0$ are adapted, predictable processes. Since we are not able to find the solution $C_t(S_t)$ in this context, we use the Black-Scholes-Merton model as a first-approximation. This model will have the same dynamics as above, but instead will have as its source of randomness a Wiener Process W_t^*. We can then denote the Black-Scholes option price in this context as $V_t(S_t)$, where according to convention we let $\Gamma_t = \frac{\partial^2 V}{\partial S^2}(S_t)$.

Now, say we hedge using the Black-Scholes model with constant volatility σ. Assuming a constant risk-free rate r, it can be shown (see Mitoulis (2019)) that the hedging profit and loss (PnL), U_t, is a stochastic process, with terminal value:

$$U_T = \frac{1}{2}\int_0^T e^{r(T-t)}\Gamma_t S_t^2(\sigma^2 - \sigma_t^2)dt. \tag{1.2}$$

Since $\left(\frac{dS_t}{S_t}\right)^2 = \sigma_t^2 dt$, for $0 = t_0, t_1, \ldots, t_n = T$, we can discretize this as:

$$U_T(\sigma) = \frac{1}{2}\sum_{i=1}^{n-1} e^{r(T-t_i)}\Gamma_i S_i^2 \left(\sigma^2(t_i - t_{i-1}) - \left(\frac{S_i - S_{i-1}}{S_{i-1}}\right)^2\right), \tag{1.3}$$

$$\text{where} \quad \Gamma_i = \frac{\Phi(d_1)}{S_i\sigma\sqrt{T-t_i}}, \qquad d_1 = \frac{\ln\left(\frac{S_i}{K}\right) + (r + \frac{1}{2}\sigma^2)(T-t_i)}{\sigma\sqrt{T-t_i}},$$

and $\Phi(.)$ is the cumulative Normal distribution function. However, in our analysis we found the Γ_i term to make the BEV results unfeasably erratic. So, instead, if we define $\Delta_i := \Phi(d_1(S_i))$ (i.e., the Black-Scholes delta hedge at time t_i) we can calculate the terminal PnL explicitly as:

$$U_T = \sum_{i=0}^{n-1}\left[(V_{i+1} - V_i) - \Delta_i(S_{i+1} - S_i) - (V_i - \Delta_i S_i)\frac{e^{-r\tau_{i+1}} - e^{-r\tau_i}}{e^{-r\tau_i}}\right], \tag{1.4}$$

where $\tau_{i+1} = t_{i+1} - t_i$ and for ease of notation we define $t_{-1} = 0$.

Dupire (2006) presents two approaches to solve for BEV. The first calculates a BEV surface for each historical period examined [1]. It is accepted practice to use non-overlapping historical periods to preserve the independence and identical distribution of these volatilities (Mitoulis, 2019). Having one BEV per sample path enables one to find the mean BEV over all historical periods, as well as allowing for calculation of pointwise quantile/percentile ranges.

Algorithm 1: Average the BEVs

```
select a model M for which the Δ-hedging process is known
generate m sample paths according to model M
forall independent historical periods (m) do
    forall options with strikes (X_i)_i and terms-to-maturity (τ_j)_j do
        calculate the values of Δ_{0≤t≤τ_j} under M
        Δ-hedge the option to maturity
        calculate Δ-hedging PnL, U_{i,j}
        express U_{i,j} as a function of volatility σ(X_i, τ_j)
        find the BEV such that U_{i,j}(σ_{i,j}(X_i, τ_j)) = 0
    end
    obtain a volatility surface V_h = {σ_{i,j}(X_i, τ_j)}_{i,j}
end
average all the surfaces {V_h}_h to obtain one BEV Surface
```

A second method which carries computational and convergence advantages, and produces smoother surfaces (Mitoulis, 2019), is to calculate the average PnL

[1] In this paper, we use 'historical period' here to describe retrospectively examining a simulated price path.

over all historical periods for a single option. The BEV is then found as the single volatility applying to all periods which sets the average PnL to zero. If this process is repeated for all options, we will obtain a BEV surface. However, we will not obtain a distribution of volatilities (which would provide a level of confidence in the estimate) (Mitoulis, 2019).

Algorithm 2: Zero the average PnL

select a model M for which the Δ-hedging process is known
generate m sample paths according to model M
forall *options with strikes* $(X_i)_i$ *and terms-to-maturity* $(\tau_j)_j$ **do**
  forall *independent historical periods* (m) **do**
    calculate the values of $\Delta_{0\leq t\leq \tau_j}$ under M
    Δ-hedge the option to maturity
    calculate Δ-hedging PnL, U_m
  end
  calculate $\bar{U}$, the average of $\{U_m\}_m$
  express $\bar{U}$ as a function of volatility $\sigma(X_i, \tau_j)$
  find the BEV such that $\bar{U}(\sigma_{i,j}(X_i, \tau_j)) = 0$
end
obtain a BEV surface $\mathcal{V} = \{\sigma_{i,j}(X_i, \tau_j)\}_{i,j}$

1.1.3 Beyond Volatility

In trying to accurately price options, we have thus far presented volatility estimation procedures (parametric and the semi-parametric approach of BEV). Using these volatility estimates as model inputs when pricing naturally carries implicit model assumptions and implications. Implementing these models when pricing means violating assumptions of continuous hedging (Halperin, 2020), and trading in markets where not all options will have a tradeable underlying: in this incomplete market setting, unique derivative prices will not exist (Hulley and McWalter, 2008). For completeness, we note Fliess and Join (2010), who propose an interesting model-free delta hedging approach using trends in financial time series, which is not our focus here.

The converse to the problem of pricing an option is how to hedge it until expiry (Hull and White, 2017). We will consider a model-free approach where derivative prices are found based on explicit discrete hedging using convex risk measures (see Föllmer and Schied, 2011, ch.4) to minimise hedging losses. The pricing problem then becomes one of finding the optimal hedging strategy. The appeal of reinforcement and deep learning models lies in their ability to also solve for an optimal delta hedge as part of the training and pricing process (see Amilon (2003),Halperin

(2020) or Buehler *et al.* (2019)).

Reinforcement Learning

Reinforcement learning algorithms find optimal strategies by dynamically observing successes and failures (see Russell and Norvig, 2009, ch.21). Q-Learning, as proposed by Watkins and Dayan (1992), is a computationally efficient reinforcement learning technique which iteratively improves judgement of the quality of actions taken in particular states. Halperin (2020) uses Q-Learning to find the optimal hedge (for a replicating portfolio of the underlying and a riskless bank account) and option price from market data under quadratic risk minimisation, while also incorporating a risk premium (which means that Halperin's results are not under the risk-neutral measure). Regarding models and the volatility smile, Halperin (2020) states that both volatility surface and jump diffusion models are unnecessary: the *only* requirement is quadratic risk minimisation.

These results are extended and elaborated again by Halperin (2019), showing that the reinforcement learning solution recovers the Black-Scholes-Merton result when risk-aversion is disregarded. Halperin shows that his QLBS model still learns the hedge when trading decisions are neither consistent nor optimal - this result is used to price portfolios of options (2019).

Halperin (2019) drives to the heart of the matter, remarking that traditional continuous-time models provide no intuition regarding the risk of the hedging strategies they produce, while the risk-return tradeoff is the main goal of hedging solutions obtained by the QLBS model.

Deep Learning

Deep learning techniques use feedforward neural networks to approximate optimal functions (strategies) using a training data set, which are then used on out-of-sample data (see Goodfellow *et al.*, 2016, ch.6). Buehler *et al.* (2019) model hedging strategies as neural networks (using various trading signals in addition to the price of hedging instruments). Termed *deep hedging* (DH), these techniques are again completely model-free in the sense that they do not rely on assumptions regarding the dynamics of the underlying stock; all that is needed to converge to an optimal strategy is some sort of cost/loss minimisation function.

Buehler *et al.* (2019) examine an incomplete market setting where option prices will not be unique. They pick Conditional-Value at Risk (CVaR) as a convex risk measure to obtain a price for a specific CVaR-significance level. The advantage of this is that instead of the unique option price returned by traditional volatility-

driven models, one can plot an array of option prices with respect to different CVaR-significance levels, which would provide a market-maker further insight into the overall price making/discovery process. The CVaR-significance level can also be seen as a proxy for risk-aversion (Kisiala, 2015).

As an illustration, Buehler *et al.* (2019) use simulated data generated by Heston's (1993) stochastic volatility model to compare their out-of-sample DH results to the unique hedge given by the model. The deep learning solution performs well, and is a close approximation to the model hedge for risk-neutral preferences.

Brannelly *et al.* (2019) implement the DH method proposed by Buehler *et al.* (2019), first simulating stock prices with constant volatility (as in the Black-Scholes-Merton framework) and comparing the optimal solution to the Black-Scholes delta. They found the neural network to perform well when the option was not close to maturity (as the function which the network was trying to learn was not complex). Brannelly *et al.* (2019) then simulated stock prices with stochastic volatility (as in the Heston (1993) framework), but with parameters which were intentionally similar to what could be expected of Black-Scholes-Merton dynamics. They found the results to reasonable. Finally, Brannelly *et al.* (2019) used an exponential utility function to calculate option prices using DH under an indifference pricing framework. The neural network prices were compared to Monte Carlo methods and did not show a large deviation.

1.2 Contribution to the Literature

Bearing the result from Crawford (2019) in mind that standard BEV is a reasonable method to use for time-homogeneous models, we first attempt to show the same for the DH approach. Secondly, we compare whether DH may in fact be a better approach to use for time-inhomogeneous models (and thus the real world) when compared to BEV. In both cases, within the controlled simulation environment which we create, we investigate and observe whether DH presents an improvement over standard BEV in recovering the model-implied volatility surfaces. So, to be explicit, this work examines whether DH has any advantage over standard BEV, particularly: We test whether the model-free non-parametric delta hedge function of a neural network poses any advantage over the Black-Scholes delta hedge function.

Thirdly, where applicable based on the previous results, we analyse the inclusion of risk aversion and compare the results between BEV and DH in this context. This augmentation is contingent on the first point producing sensible results (i.e. DH presenting a viable alternative to standard BEV) which would warrant the further analysis here.

1.3 Dissertation Outline

In what follows, we will first describe in Chapter 2 the market setting used to form the optimisation problem for our DH neural network, as well as the various models used for stock price simulation. Thereafter, we will define and explain the architecture of our DHNN in detail in Chapter 3. In Chapter 4 we present a risk-neutral setting, in which we verify that our DHNN can recover the true Black-Scholes delta hedge, as well as comparing BEV with the implied volatility surfaces of our DHNN and whether both approaches can recover the model-implied volatility surfaces. Thereafter, in Chapter 5, we extend our analysis to a risk-averse setting, using the CVaR convex risk measure for both BEV and our DHNN. We investigate how CVaR skews our hedging PnL, the effects of varying risk aversion, and how the volatility skews differ for BEV and the DHNN. Chapter 6 concludes.

Chapter 2

Market Setting

First we introduce the environment in which the stock prices will be simulated and arrive at a general form of the optimisation problem which the Deep Hedging neural network (DHNN) will attempt to solve in equation (2.4). We proceed to then describe and justify the use of convex risk measures needed to constrain (2.4). Lastly, we describe the dynamics according to which the simulated stock prices will evolve.

2.1 Forming the Optimisation Problem

Consider a discrete-time setting $0 = t_0 < t_1 < \cdots < t_n = T$ in which we have written, at t_0, an at-the-money (strike X) European call option on a stock (which pays no dividends) with maturity T. Further, assume that time steps are business days and of equal size (i.e., $t_{k+1} - t_k = \mathrm{d}t \ \forall \ k$). If we set $n = 63$ trading days this will give our option a maturity of roughly three months. To simplify matters we assume that the risk-free interest rate, r, is zero.

Assume that in this market there is only one security, the stock itself: a stochastic process $S = (S_k)_{k=0}^n$ where S_k is the stock price at time t_k. Let $\mathbb{F} = (\mathcal{F}_k)_{k=0}^n$ be the filtration generated by S. Each $\mathcal{F}_k$ will thus represent all available information at time t_k.

Remark 2.1. Denote the option payoff as the $\mathcal{F}_T$-measurable random variable Z. If we assume that this market is arbitrage-free then by the Fundamental Theorems of Asset Pricing there must then exist at least one Equivalent Martingale Measure. Since Z is $\mathcal{F}_T$-measurable, we know from the Martingale Representation Theorem that the existence of a replicating portfolio for this liability is guaranteed.

To hedge the liability, $-Z$, defined by the option payoff at time T we use a hedging strategy $\delta = (\delta_k)_{k=0}^{n-1}$ (also an $\mathbb{F}$-adapted stochastic process), where $\delta_k(S_k)$ denotes our stock holdings from time t_k to t_{k+1}. Define $\mathcal{H}$ as the set of possible hedg-

ing strategies. Assuming no transaction costs, we can then represent our hedging profit and loss (PnL) at time T as:

$$(\delta \cdot S)_T := \sum_{k=0}^{n-1} \delta_k (S_{k+1} - S_k). \tag{2.1}$$

We will then seek to charge a premium of p for the option such that we can meet the terminal value of the option liability:

$$-p(Z) + (\delta \cdot S)_T = Z. \tag{2.2}$$

Or, alternatively, we would like the value of our terminal portfolio to be (at least) exactly offset by the premium we receive for writing the option:

$$p(Z) = -Z + (\delta \cdot S)_T. \tag{2.3}$$

We can then define the optimization problem which our DHNN faces as:

$$\pi(-Z) := \inf_{\delta \in \mathcal{H}} \rho(-Z + (\delta \cdot S)_T) \tag{2.4}$$

where ρ is an appropriate cost function used to train the DHNN. Naturally, the optimal hedging strategy δ^* will be the δ which minimises (2.4).The price which our DHNN arrives at will then be:

$$p^*(Z) = \pi(-Z) - \pi(0), \tag{2.5}$$

an indifference price which will hold if π is translation invariant. When ρ is convex, π is indeed translation invariant (Buehler *et al.*, 2019). Alternatively, we can arrive at the price p^* by retrospectively using the optimal hedge δ^*:

$$p^*(Z) = -Z + (\delta^* \cdot S)_T. \tag{2.6}$$

In our analysis, we follow Buehler *et al.* (2019) and choose to focus on finding optimal option prices (and hedging strategies) using *convex risk measures* for ρ. Convex risk measures carry helpful intuition described in Definition 2.2 below.

Definition 2.2. Let Y be the time T hedging return (or, 'error'). Using 2.3, we define $Y := -Z + (\delta \cdot S)_T$. Let $\mathcal{Y}$ be the set of all possible returns in this market (so losses are permitted). For $Y, Y_1, Y_2 \in \mathcal{Y}$, a *convex risk measure* $\rho : \mathcal{Y} \to \mathbb{R}$ is:

- *Cash-invariant*: $\rho(Y + c) = \rho(Y) - c$.

  Adding cash to a position reduces the risk of that position by the same amount.

- *Monotonically decreasing*: if $Y_1 \geq Y_2$ then $\rho(Y_1) \leq \rho(Y_2)$.

  So the bigger our loss, $-Y$, the bigger our risk.

- *Convex*: $\rho(\alpha Y_1 + (1-\alpha)Y_2) \leq \alpha\rho(Y_1) + (1-\alpha)\rho(Y_2)$, where $\alpha \in [0,1]$.

  Diversification reduces risk.

We will use two convex risk measures in particular: Mean Squared-Error (MSE), and Conditional Value-at-Risk (CVaR) (as proved in Rockafellar and Uryasev (2000)). MSE is 'risk-neutral' in the sense that CVaR incorporates the risk-appetite of the market-maker. It is for this reason that MSE will be used to determine under which stock price regimes it is feasible to examine the performance of the DHNN with a CVaR cost function. But, to do this, we must first introduce the different regimes used for simulation.

2.2 Model Simulation

Below we will justify and describe the use of three different stock price simulation procedures which will be used in our analysis. A list of market conventions and parameters used is included in Appendix A.

2.2.1 Justification

We choose to use simulated price paths in our analysis for three different reasons:

1. *Model verification*: simulating price paths which follow geometric Brownian motion allows us to verify that the model does indeed recover the optimal Black-Scholes option price at t_0 and delta hedge values at each time step $(t_i)_{i=0}^{n-1}$.

2. *Regime changes*: in the incomplete market setting, we have the freedom to change model structures and specifications and see how the DH neural network performs and reacts to different stock price dynamics in a controlled way.

3. *Comparable analysis*: simulated data allows us the benefit of understanding analytical solutions for option prices (and the implied volatility surfaces which follow) from the model used for simulation. There is no basis for comparison with real world data as the underlying data generating process/model is unknown.

However, to make our analysis more realistic, we simulate an environment where one would have to resample from one stock price path stretching back into the past, explicitly described in Algorithm 3 below:

Algorithm 3: Simulate overlapping stock price paths

take an option with k time steps to maturity
select a model M for the underlying
select a reasonably observable historic number of time steps n
generate one sample path of n time steps according to model M
then, starting at the first time step $j = 0$:
forall *timesteps* j **do**
  take a subset of size k from the sample path
  rebase the subset by multiplying each stock price by $\frac{S_0}{S_j}$
  shift forward one time step
end
return $n - k + 1$ overlapping stock price paths

Note that Algorithm 3 requires that $\left(\frac{S_0}{S_j} : j = m + 1 : m + k\right)$ is also a valid stock path for the model M. This holds for the log-type models which we are examining here, but not necessarily for other models, such as Local Volatility Models.

Realistically we would desire $k \ll n$ in order to generate as many (albeit overlapping) sample paths as possible. Needing to use overlapping sample paths presents a situation more akin to real-life data constraints, but poses an additional challenge to the predictive capabilities of our DHNN as the stock price paths will no longer be independent.

2.2.2 Different Models Used

To conclude the chapter, we describe below the three different stock price simulation models which we use in our analysis.

Geometric Brownian Motion

This is the simplest case, for which dynamics are relatively unrealistic and which is thus used as a sanity-check for the effectiveness of the DHNN. Let μ be the drift of the process, σ the variance, and W_t a Wiener process, then the stock price will evolve according to:

$$\mathrm{d}S_t = S_t \left(\mu \mathrm{d}t + \sigma \mathrm{d}W_t\right) \tag{2.7}$$

See Algorithm 4 for the simulation procedure.

Merton's Jump Diffusion Model

The second regime, the Merton jump diffusion model, has dynamics:

$$\mathrm{d}S_t = S_t \left(\mu \mathrm{d}t + \sigma \mathrm{d}W_t + J \mathrm{d}N_t\right) \tag{2.8}$$

with μ, σ and W_t the same as for geometric Brownian motion. The difference here is the inclusion of random jumps: $J \sim \mathrm{N}(\mu_j, \sigma_j^2)$. N_t is a homogeneous Poisson process where $\mathbb{P}(N(t) = k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$ and λ is the average number of jumps per unit time. See Algorithm 5 for the simulation procedure.

Heston Model

The last model we use is the Heston model, which has dynamics:

$$\mathrm{d}S_t = \mu S_t \mathrm{d}t + \sqrt{\nu_t} S_t \mathrm{d}W_t^{(1)} \tag{2.9}$$

$$\mathrm{d}\nu_t = \kappa(\theta - \nu_t)\mathrm{d}t + \sigma\sqrt{\nu_t}\mathrm{d}W_t^{(2)} \tag{2.10}$$

where μ is as above, ν_t is the instantaneous stock price variance, θ is the long-run stock price variance, κ is the rate at which ν_t returns to θ and σ is the variance of variance. $W_t^{(1)}$ and $W_t^{(2)}$ are Wiener processes as before, with correlation ρ. See Algorithm 6 for the simulation procedure.

Chapter 3

The Deep Hedging Neural Network

We begin this chapter by justifying why it is possible for a Deep Hedging Neural Network (DHNN) to approximate the optimal hedging strategy (and thus find the minimum option premium we would be willing to charge). We then expound upon the structure of our neural network in particular, and end with a discussion on the implementation of, and hyperparameters chosen for, our network.

3.1 Justification

The Universal Approximation Theorem explored by Hornik *et al.* (1990) shows that feed-forward networks, under certain conditions, may approximate multivariate functions to an arbitrary level of precision, even with just one hidden layer. All that is required is for the activation function to be smooth, and the function being approximated to have a generalised derivative (as some piecewise differentiable functions do). For these feed-forward networks, Scarselli and Chung Tsoi (1998) tackle the computational aspect of achieving accuracy within a desired degree of approximation (and show how the architecture of the neural network can be constructed to achieve this).

Buehler *et al.* (2019) choose a bounded, non-constant activation function and use the aforementioned Universal Approximation result to frame the broad question of finding an optimal hedging strategy instead as a finite-dimensional problem constrained by finding optimal neural network parameters. Following their approach (and notation where possible) we can then define our feed-forward neural network in general terms:

Definition 3.1. Let $L, N_0, N_1, \ldots, N_L \in \mathbb{N}^+$ and $l = 1, \ldots, L$ for a feed-forward neural network with input dimension N_0, output dimension N_L and hidden layer dimensions $N_1, \ldots, N_{L-1}$ respectively. Furthermore, define $\sigma_l : \mathbb{R} \to \mathbb{R}$ as a bounded,

non-constant activation function and let $W_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ be an affine function of the form $W_l(s) = A^l s + b^l$. We represent this network $F : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ as:

$$F(s) = W_L \circ f_{L-1} \circ \cdots \circ f_1 \tag{3.1}$$

$$\text{where} \quad f_l = \sigma_l \circ W_l \quad \text{for} \quad l = 1, \ldots, L-1 \tag{3.2}$$

We then denote the set of parameterisation values for the matrices $A^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b^l \in \mathbb{R}^{N_l}$ for all layers as θ.

Now, recall that in our market setting delta is only a function of stock price. So, to find the optimal $\delta_k(S_k)$ at each time step, we must create a neural network at each time step, $F_k = F_k(S_k; \theta_k)$, which takes the stock price at that time step as its input. Again, to repeat the result of Buehler *et al.* (2019), our problem has now been rephrased from one of finding the optimal hedging deltas $\delta = (\delta_k)_{k=0}^{n-1}$ to one of finding the optimal parameter set $\theta = (\theta_k)_{k=0}^{n-1}$ from the set of possible parameters Θ.

To be explicit regarding the above, at each time step we aim to find a set of parameters θ_k such that:

$$F_k(S_k; \theta_k) = \delta_k^*. \tag{3.3}$$

This would mean that we could represent our hedging profit and loss (PnL) described in (2.1) as

$$(\delta^* \cdot S)_T = \sum_{k=0}^{n-1} F_k \cdot (S_{k+1} - S_k). \tag{3.4}$$

When we have found the optimal parameter set θ, we can combine (3.4) with the value of our terminal liability (in a similar manner to (2.3)) to infer the minimum option price which our DHNN is willing to charge:

$$p^*(Z) = -Z + \sum_{k=0}^{n-1} F_k \cdot (S_{k+1} - S_k). \tag{3.5}$$

However, in order to do this, we must first train our DHNN on a constrained version of (3.5) using a convex risk measure as in Definition 2.2. We can thus represent our final minimum training loss as:

$$\pi(-Z) = \inf_{\theta \in \Theta} \rho \left(-Z + \sum_{k=0}^{n-1} F_k \cdot (S_{k+1} - S_k) \right). \tag{3.6}$$

Crucially, we know by Proposition 4.9 in Buehler *et al.* (2019) that our optimal DHNN option price $p^*(Z) = \pi(-Z) - \pi(0)$ will converge to to the exact price $p(Z)$.

Note here once again the explicit distinction between (2.4) and (3.6) which follows from the Buehler *et al.* (2019) result: we have moved from an infinite-dimensional problem of finding the optimal hedge at every timestep (or, infinite state space functions for the hedge parameters) toward a constrained, finite-dimensional problem of finding the parameter set θ which minimises the value of our terminal portfolio of hedging PnL and option payoff liability (or, functions that are parameterised by a finite number of parameters but are still non-parametric).

3.2 Construction

We will implement n simple feed-forward neural networks $F_k, k = 0, \ldots, n-1$, with two hidden layers each. For our hidden layers, we pick $\tanh$ (bounded, non-constant) activation functions and arbitrarily use 32 nodes. Importantly, we use a sigmoid activation function in the final layer to ensure that our hedge output $\delta_k^* \in [0, 1]$. So, to be consistent with our notation above, each F_k will have:

Description	Variable	Value
Layers	L	3
Input dimension	N_0	1
Hidden layer dimension	N_1, N_2	32
Hidden layer activation functions	$\sigma_1, \sigma_2 : \tanh$	$\sigma_l(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Output dimension	N_3	1
Output layer activation function	$\sigma_3 :$ sigmoid	$\sigma_l(x) = \frac{1}{1+e^{-x}}$

Table 3.1: Feed-forward neural network architecture for F_k

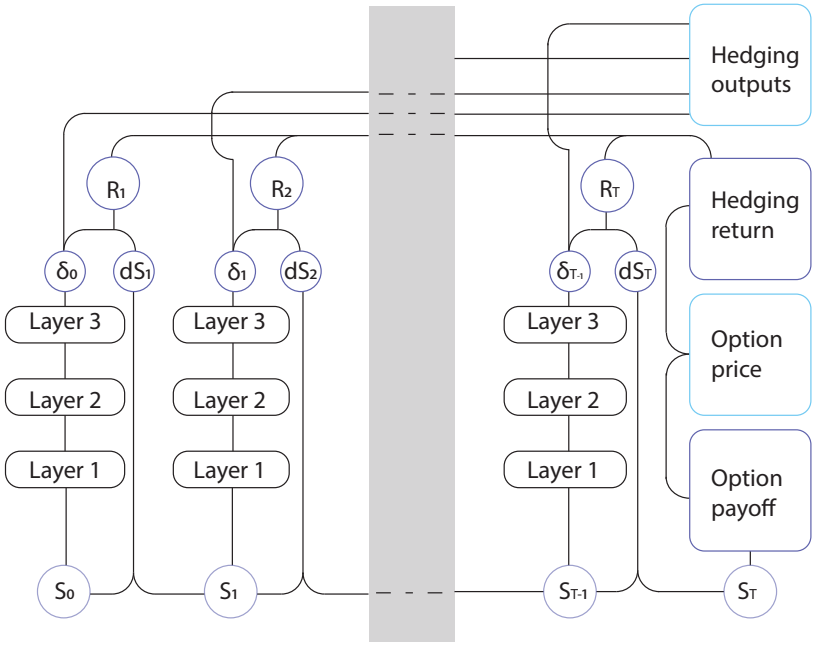
The model architecture is illustrated in Figure 3.1 below [1].

At each time step t_k, the current stock price S_k is fed into the model. The hedging output from the final layer, δ_k^* (i.e., the result of implementing (3.3)), is then multiplied by the stock price change between t_k and t_{k+1}, $\mathrm{d}S_{k+1}$. Note that this stock price change is not in percentage terms, and so $R_{k+1} = \delta_k^* \mathrm{d}S_{k+1}$ represents the monetary gain from holding δ_k^* units of stock from time t_k to t_{k+1}. If we take $\sum_{k=1}^{n} R_k$ (HEDGE RETURN in Figure 3.1, and the implementation of (3.4)) we will get the profit and loss from our hedging strategy, which is equivalent to (2.1).

Our concern lies in whether this hedge return will be able to meet our liability $-Z$ at time T. So at time T we calculate the liability owing (OPTION PAYOFF in Figure 3.1) to the client: $-Z = -(S_T - X)^+$. The shortfall between our hedging

[1] Dark blue borders indicate inputs, purple borders indicate intermediate calculations, and light blue borders indicate final outputs.

Figure 3.1: Deep Hedging Model Architecture



PnL and the liability owing will be the final option price (OPTION PRICE in Figure 3.1, and the implementation of (3.5)), and can be seen as an 'error' which, once constrained by our convex risk measure, will be the model loss which we aim to minimise.

However, since by construction our model output will be the OPTION PRICE *constrained by the convex risk measure*, we still require the sequence of deltas, $\delta = (\delta_k)_{k=0}^{n-1}$, to infer our option price as in (2.3).This sequence of values is returned via HEDGE OUTPUTS above. Note that these outputs are not used in the cost function when training the model and so, once the model is trained, will allow us to infer the delta parameters for different stock price paths (and thus levels of moneyness).

This model which we have constructed is rudimentary *by design*: if a neural network as basic as this is able to recover Black-Scholes results and price comparably with traditional techniques (like Break-Even Volatility (BEV)) in a 'model-free' world it, firstly, bodes well for more powerful techniques such as increased layers, convolutions (Nielsen, 2015, Chapter 6), recurrent neural nets (Medsker and Jain, 2001) and LSTM models (Greff *et al.*, 2016). Secondly, it charts a course which allows us to compare our DHNN with BEV techniques in a generalised setting of convex risk measures.

3.3 Implementation

We implement the DHNN (as well as all other analysis) in Python, making extensive use in particular of the Keras (2015) Deep Learning API and choosing to use the TensorFlow (2015) computation backend. See Géron (2019) for an introduction.

We initialise our network parameters randomly using the truncated Glorot normal distribution. When training our DHNN, we use the Adam optimisation introduced by Kingma and Ba (2014), an adaptive learning rate algorithm which is cognisant of exponentially decaying past gradients (and their squares) and thus requires relatively little hyperparameter tuning (Géron, 2019). In Keras, we use the default Adam parameters, including a learning rate of 0.001. We find Adam to be sufficient for our purposes in preventing exploding gradients as discussed in (Philipp *et al.*, 2018) and (Hanin, 2018).

The problem of what batch size to select has been extensively debated (see (Keskar *et al.*, 2016) or (Devarakonda *et al.*, 2017)), with proponents for small (see Masters and Luschi (2018)) and large batches (see Hoffer *et al.* (2018) or Smith *et al.* (2017)) to be found in the literature. Hesitant to be drawn in to an involved exercise of investigating optimal hyperparameters for the model, we choose an arbitrary batch size of 10% of the training sample size and train for 1000 epochs.

Chapter 4

Risk-Neutral Setting

In this chapter, we will introduce and justify the use of the Mean Squared-Error risk measure and in (4.2) present explicitly how it is used in the training of our Deep Hedging Neural Network (DHNN). Thereafter, we will use the network results in a complete market setting as a proof-of-concept for the DHNN, going on to analyse, critique, and suggest improvements for the DHNN performance against Break Even Volatility (BEV) in an incomplete market setting.

4.1 Risk-Neutral Optimisation Problem

We know from Remark 2.1 that the existence of a replicating portfolio for our terminal liability is guaranteed. Thus, Equation (2.2) shows us that this implies the existence of a hedging strategy δ (though not necessarily unique). Within the framework developed by Föllmer and Sondermann (1985) and Schweizer (1991), we say that our hedging strategy will be *admissible*. We thus choose the mean squared-error (MSE) cost function to train our network since the δ^* which minimises (2.4) under the MSE convex risk measure will be equal to one such admissible strategy, δ.

Define $Z^{(j)}$ and $S^{(j)}$ to be the liability and stock price path for the j^{th} simulation respectively. Now, if we explicitly formulate our convex risk measure ρ in the optimisation problem (2.4) as an MSE cost function, we may rewrite the equation as:[1]

$$\pi(-Z) = \inf_{\delta \in \mathcal{H}} \left[\frac{1}{m} \sum_{j=1}^{m} \left(-Z^{(j)} + (\delta \cdot S^{(j)})_T \right)^2 \right], \tag{4.1}$$

[1] We are actually taking the MSE between our terminal portfolio value and 0, i.e., forcing the value to be as close to 0 as possible. Thus p^* will be the minimum price which we are willing to receive for writing the option.

summing over each simulation j in our training set of size m. Using the justification expounded upon in Chapter 3, and particularly (3.6), we may rewrite (4.1) as:

$$\pi(-Z) = \inf_{\theta \in \Theta} \left[\frac{1}{m} \sum_{j=1}^{m} \left(-Z^{(j)} + (F \cdot S^{(j)})_T \right)^2 \right], \tag{4.2}$$

the explicit optimisation problem which we will use to train our Deep Hedging Neural Network (DHNN) in the risk-neutral setting. Recall that $F_k(S_k; \theta_k) = \delta_k^*(S_k)$ and so we may then use the hedging parameter outputs of our DHNN, δ^*, to infer the minimum option price to charge, p^*, determined by our network using (2.6):

$$p^* = \frac{1}{m} \sum_{j=1}^{m} \left(Z^{(j)} - (\delta^* \cdot S^{(j)})_T \right). \tag{4.3}$$

Note that in this case it does make sense to only arrive at an option price once we have evaluated the entire evolution of the price path, because we would like to arrive at a best estimate of the option price based on the underlying data which we have used to train our model. This approach is also analogous to BEV in the sense of retrospectively analysing historical or simulated data.

4.2 Complete Market

First, we consider a complete market in a Black-Scholes world, where just one instrument (plus a cash bank account with interest rate $r = 0$) will be sufficient to hedge the option payoff: the underlying stock itself, since stock prices (which evolve according to geometric Brownian motion) depend on one source of randomness only. Since our liability $-Z$ will be replicable using only a combination of the instruments at hand (i.e., the stock and a bank account), it is thus attainable, which means we have a complete market setting.

The advantage of the model in also returning the hedging strategy δ_k^* at each time step will allow us to observe whether the DHNN is able to converge toward the known (see, for example, (Hull, 2015, pg.404)) closed-form Black-Scholes hedging strategy:

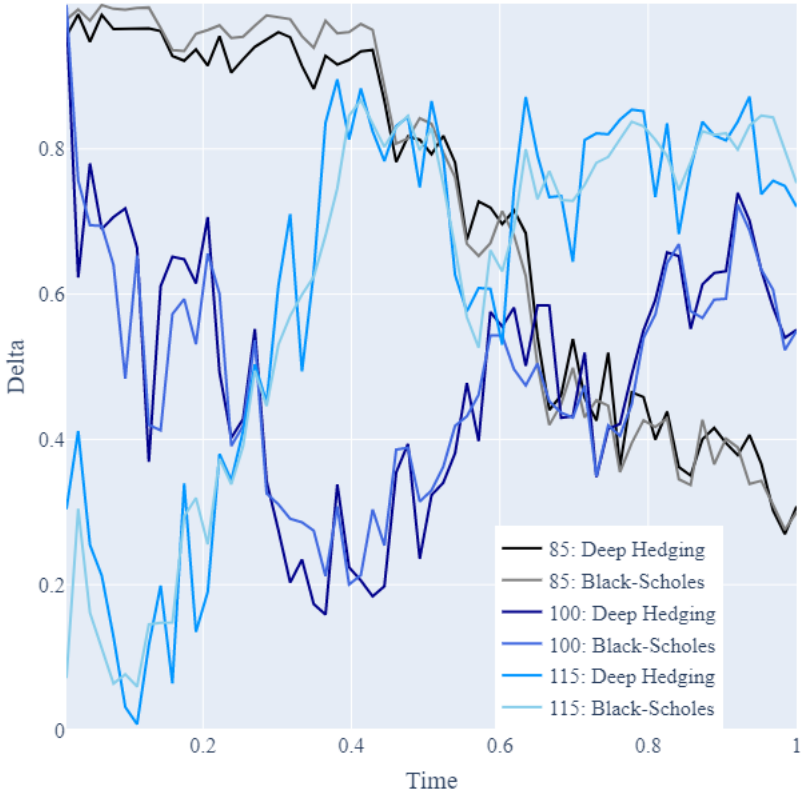
$$\delta_k^{\text{BS}}(S_k) = \Phi \left[\frac{\ln\left(\frac{S_k}{X}\right) + \left(r + \frac{1}{2}\sigma^2\right)(T - t_k)}{\sigma\sqrt{T - t_k}} \right] \tag{4.4}$$

where $\Phi(.)$ is the cumulative Normal distribution function as before and σ is the stock price volatility.

So in pursuit of this convergence result, instead of using overlapping price paths, in this case we train our DHNN instead on 100 000 independent simulated

sample paths in batches of 10 000 for 1 000 epochs. Below in Figure 4.1 we plot our DHNN delta outputs $(\delta_k^*)_{k=0}^{n-1}$ for an illustrative sample of three simulated paths (an out-the-money, at-the-money and in-the-money case of $S_0 = 85, 100, 115$ respectively for a strike $X = 100$) which evolve through time from option contract inception until one time step before maturity. On the same set of axes we plot the *true* Black-Scholes delta result $(\delta_k^{\text{BS}})_{k=0}^{n-1}$. For a model with as simple an architecture as that described in Figure 3.1, the accuracy is remarkable.

Figure 4.1: Black-Scholes Delta Recovery



However, in a more realistic setting, one is less likely to have access to such a rich set of training data. For our remaining analysis we will use simulated price paths sampled from overlapping intervals as described in Algorithm 3. Now, if we subtract $p(Z)$ from both sides of the equation, we can rearrange (2.3) as:

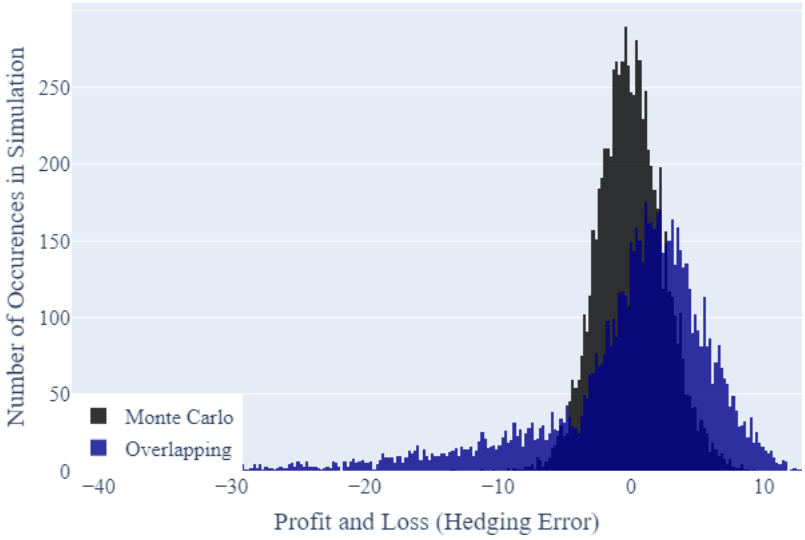
$$G_j = -Z^{(j)} + (\delta^* \cdot S^{(j)})_T - p^* \tag{4.5}$$

to denote G_j as our *portfolio gain or loss* at maturity for a specific simulated path j. In other words, for the specific price path j, what would be the change from the initial zero self-financing value of our portfolio, assuming that we sold the option for p^* at t_0, hedged according to δ^* and fulfilled our obligation Z at maturity. As a

market maker, we would prefer G to be as small as possible since it is unpredictable. When training on independent sample paths, an MSE cost function will allow for residual error (i.e., G) which is symmetrically distributed around zero. In other words, market makers will be *neutral* between portfolio gains or losses, or, would want the PnL to have zero mean and as small a variance as possible - theoretically, in the complete market setting, it is possible make this distribution degenerate, i.e., zero variance.

To explicitly view the added difficulty which our DHNN will face when training on overlapping paths we plot a histogram of our portfolio gains $G = (G_j)_{j=1}^m$ for both Monte Carlo and overlapping price paths in Figure 4.2 below. We assume 30 years' worth of historical price data and 250 trading days in a year, which gives us 7438 overlapping stock price paths, whose results we compare with those of 7438 Monte Carlo price paths' worth of training.

Figure 4.2: PnL Comparison Between Different Simulation Methods



Firstly, one can note the symmetrical distribution of G around zero for Monte Carlo price paths. Secondly, one can note both the wider distribution and longer left tail of the portfolio gains and losses for the overlapping price paths. What this means is that when training on overlapping price paths, the DHNN finds it relatively more difficult to recover a hedging strategy $\delta^* = \delta$, and concerningly for us as market makers, portfolio losses will likely be larger than gains at maturity.

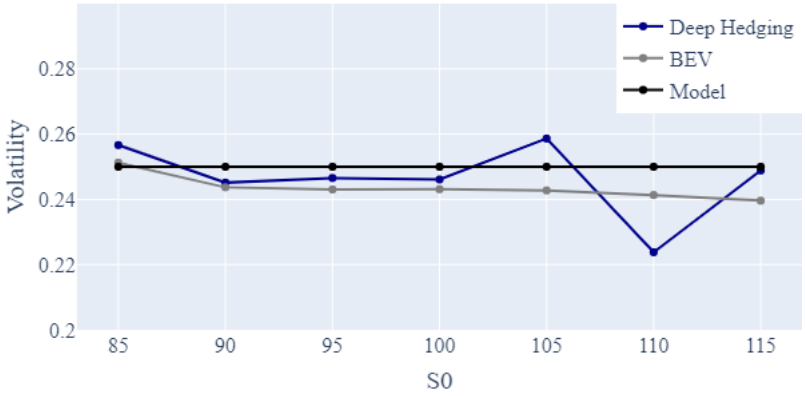
With the preliminary analysis out of the way, we may now begin our comparison of Break-Even Volatility (BEV) and our DHNN. First, in this complete market setting we investigate the pricing accuracy of both approaches, as compared with the known closed-form Black-Scholes option price solution.

We use Algorithm 2 to calculate BEV, solving the optimisation problem:

$$\sigma^{\text{BE}} = \underset{\sigma \in \mathbb{R}^+}{\arg\min} \left| \bar{U}(\sigma) \right|, \tag{4.6}$$

where we use the discretised expression for profit and loss as in (1.4). For the Deep Hedging case we solve for the volatility which sets the Black-Scholes price equal to p^*. In Figure 4.3 below, we plot the volatilities of both approaches for different levels of moneyness. We can see that both recover the true constant model-implied volatility of $\sigma = 0.25$ reasonably well, although the use of overlapping paths in both cases has made for considerably more variability around σ than would be the case with Monte Carlo paths.

Figure 4.3: Volatility Surfaces Under the GBM Model Using MSE

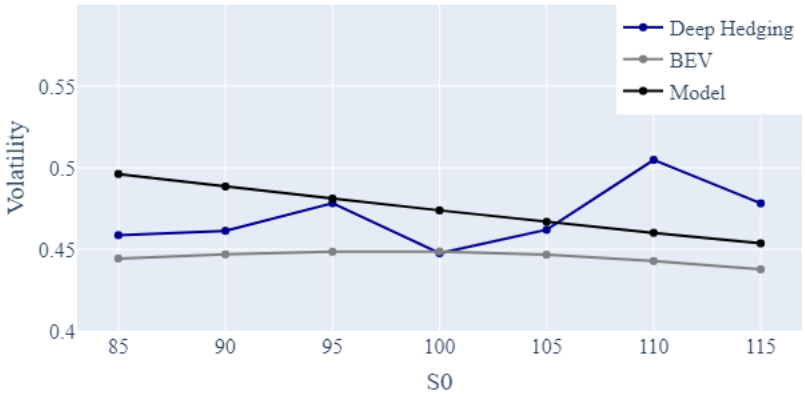


4.3 Incomplete Market

Next, we would like to train and test the DHNN in a pseudo-real world setting: an incomplete market where the liability $-Z$ will not be replicable using just a bank account and the stock itself. This is achieved by using models for the underlying stock price which permit multiple sources of randomness. This approach will be useful in allowing us to compare the model-independent implied volatility from the DHNN option price to the BEV method common in practice in a close-to real world setting.

Recall from (1.4) that in our BEV implementation we assume Black-Scholes dynamics for pricing and hedging the option, so now that we have moved away from an environment of GBM price paths the BEV approach is merely an approximation. We again plot an implied volatility surface for both the BEV and DHNN approaches but this time for Merton price paths (with parameters detailed in Appendix A).

Figure 4.4: Volatility Surfaces Under the Merton Jump-Diffusion Model Using MSE

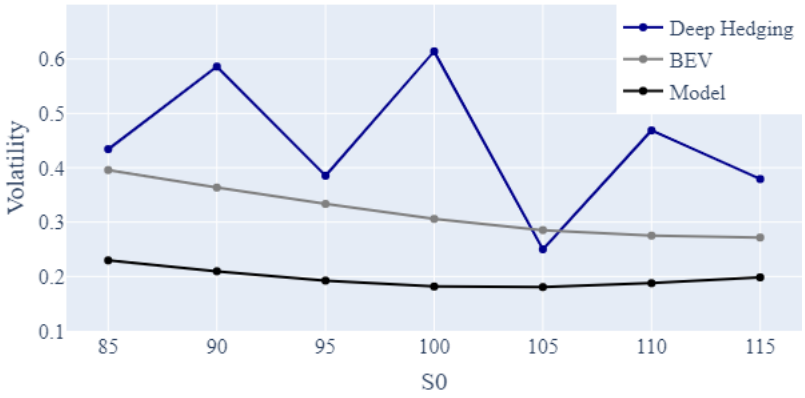


Notice that both approaches again fall within roughly 10% of the true model implied volatility. One should bear in mind that our implementation of BEV will produce more stable estimates by construction, since we are optimising over the entire sample (i.e., the volatility which sets the *mean* profit and loss to zero). In the DHNN case, we back out a price from δ^* which we use to obtain an implied volatility. Given the issues highlighted above regarding PnL distributions in Figure 4.2 for overlapping paths, we can expect a more erratic price for the size of our training data. Taking this into consideration, the performance of the DHNN against BEV is promising.

The final comparison we make is within a Heston model environment, where stock prices evolve with stochastic volatility. The Heston parameters used are also given in Appendix A. We find the DHNN results in Figure 4.5 below to be far inferior to the BEV case. Notice again that as a direct result of the smoothing effect of zeroing the mean PnL which Algorithm 2 uses, the BEV under the Heston regime recovers the volatility skew relatively well. However, the magnitude of the difference between BEV and the true model volatility surface is still concerning, with BEV significantly overestimating the Black-Scholes implied volatility of the underlying price paths.

The more egregious error, meanwhile, lies with our model-free neural network approach. In this environment, our simple feedforward DHNN breaks down, struggling to recover the hedging δ accurately enough to be able to price the option in a range which would imply a volatility consistently near that of the true model implied Black-Scholes volatility. Our volatility estimate is also far more erratic. These shortfalls are a direct result of training the model on overlapping price paths: one can note the performance improvement of the DHNN when training on Monte Carlo paths in Appendix C. The poor performance of our DHNN when imple-

Figure 4.5: Volatility Surfaces Under the Heston Model Using MSE



mented in a Heston model environment leads us to exclude further analysis of our network in a Heston setting in the risk-averse augmentations which follow in Chapter 5. Even if we had sufficient Heston data such that we were able to sample non-overlapping sub-paths, the results would still be poor because the Heston model is time-inhomogeneous.

4.4 Residual Risk

What is clear from the above analysis (particularly Figures 4.2 and 4.5) is that, when using the MSE convex risk measure, the case for a DHNN with performance comparable to or better than BEV can only realistically occur in a Monte Carlo simulation setting. This means that one needs to take care when applying the DHNN:

(a) One should either ensure that one has a rich enough data set of independent sample paths on which to train the network, since using a Monte Carlo simulation precludes the problem of whether the data generating process is time-homogeneous. Or,

(b) Use a risk measure which accounts more adequately for the residual risk which dependent price paths pose to the predictive capacity of the network - particularly regarding the way such a risk measure could account for price paths which disproportionately skew the hedging PnL.

However, regarding (a), Wiese *et al.* (2019) recreate the Monte Carlo setting by using Generative Adversarial Networks (GANs) to simulate sets of independent price paths upon which a DHNN can be trained. GANs are not explored here, but present a promising avenue for further research.

Before considering (b), we note that the implementation of BEV is also challenging beyond the Black-Scholes world of geometric Brownian motion stock price paths. BEV assumes that one can use the Black-Scholes framework both to price the option liability to the market maker, and to create the equivalent stock and cash hedge portfolio holding using the Black-Scholes delta. Taking this into account, we would like to take an approach to (b) which will improve the performance of our DHNN and at the same time can account for the residual risk not captured by the Black-Scholes assumptions contained within BEV.

One option would be to use the *percentiles* of hedging PnL:

- Only calculate BEV based on the value of σ which sets the $\alpha \times 100^{\text{th}}$ percentile of PnL equal to zero, and
- Use the MSE risk measure to train our DHNN based only on hedging losses less than the $\alpha \times 100^{\text{th}}$ percentile of PnL,

which would be the equivalent of a Value-at-Risk formulation. Instead, for reasons expounded upon in Chapter 5, we will cater for residual risk not accounted for by BEV and the DHNN by using a different convex risk measure: Conditional Value-at-Risk.

Chapter 5

Risk-Averse Setting

In this chapter, we go beyond the risk-neutral setting of Chapter 4 and introduce a new convex risk measure: Conditional Value-at-Risk (CVaR), which will allow us to account for the residual hedging risk posed by training on overlapping price paths and using models which exhibit multiple sources of randomness. Similar to Figure 4.2, we again plot the distribution of Deep Hedging Neural Network (DHNN) gains, $(G_j)_j$, but this time once the model has been trained using the CVaR cost function in (5.5). Thereafter, in (5.7), we generalise Break-Even Volatility (BEV) to also incorporate CVaR in its zeroing of profit and loss (PnL) to facilitate a comparison with our DHNN. We can then plot the implied volatility of both approaches while varying our desired level of risk aversion to gain an intuition for how both approaches fare in this risk-averse setting. Lastly, to conclude our analysis, we compare the pricing results between BEV and our DHNN in this risk-averse setting.

5.1 Conditional Value-at-Risk

We start by explicitly defining Value-at-Risk (VaR) (the risk measure upon which CVaR is based) as alluded to at the end of Chapter 4.

Definition 5.1. Let Y be as in Definition 2.2. The VaR of Y is defined as:

$$\text{VaR}_\alpha(Y) := -\inf\{y \in \mathbb{R} : F_Y(y) > \alpha\}, \tag{5.1}$$

where $\alpha \in [0, 1]$ and $F_Y(y)$ is the cumulative distribution function of Y.

Bearing in mind that y will be a negative number, one can interpret VaR as the 'greatest' loss y such that $\mathbb{P}(Y \geq y) \geq 1 - \alpha$, or intuitively, we would not expect to lose more than y more than $(\alpha \times 100)\%$ of the time. However, VaR is not a coherent risk measure in the sense defined by Rockafellar and Uryasev (2000), and as shown by Yamai and Yoshiba (2005), for example, fails to take in to account any losses beyond the VaR threshold. This is concerning from our perspective as a

market maker, since one large loss could be extremely consequential. We thus turn to CVaR, a risk measure which explicitly focuses on this tail risk.

Definition 5.2. Continuing with Y as in Definition 2.2, the CVaR of Y is defined as:

$$\text{CVaR}_\alpha(Y) := -\frac{1}{\alpha}\int_0^\alpha \text{VaR}_\gamma(Y)\text{d}\gamma, \quad \alpha \in [0,1]. \tag{5.2}$$

Given that we have lost at least y (or, alternatively, that our $\text{VaR}_\alpha(Y) = -y$), CVaR can be interpreted as the amount, on average, which we would then expect to lose (i.e., conditional on losses being below $-\text{VaR}_\alpha(Y)$). Minimising CVaR effectively minimises large losses in the left tail of the empirical distribution of the portfolio Y, which is in-line with the aims of a market-maker. However, an aspect of subjectivity is introduced where the chosen siginifcance level α will reflect the relative risk-appetite of that specific market-maker.

We choose Conditional Value-at-Risk (CVaR) as our convex risk measure (and thus appropriate cost function used to train our model in 2.4) for four reasons:

- CVaR is not only convex but also coherent (see Rockafellar and Uryasev (2000) for a proof), a stronger condition.
- CVaR focuses on the tail risk introduced by disproportionately large losses.
- CVaR carries natural financial intuition behind it (Kisiala, 2015), and many optimisation techniques have been explored for the calculation itself as a direct result of this (Rockafellar and Uryasev, 2000). Finally,
- Since CVaR is also used by Buehler *et al.* (2019), this allows a direct comparison with their results.

The first two points are where the appeal of CVaR over standard VaR arises (Pflug, 2000). Now, using CVaR as our convex risk measure, the optimisation problem in (2.4) can be written as:

$$\pi(-Z) = \inf_{\delta\in\mathcal{H}} \text{CVaR}_\alpha\left(-Z + (\delta \cdot S)_T\right). \tag{5.3}$$

Now, let Y_α be the $(\alpha \times 100)^{\text{th}}$ percentile of these terminal portfolio values, and n_α the number of simulated terminal portfolio values which lie in the $(\alpha \times 100)^{\text{th}}$ percentile of all terminal portfolio values (i.e., where $Y < Y_\alpha$). Finally, let $\mathbb{I}$ be an indicator function. We can then discretise (5.2) in our market setting as:

$$\pi(-Z) = \inf_{\delta\in\mathcal{H}}\left[-\frac{1}{n_\alpha}\sum_{j=1}^{m} Y^{(j)} \cdot \mathbb{I}_{\{Y^{(j)}<Y^\alpha\}}\right], \tag{5.4}$$

where $Y^{(j)}$ is the terminal portfolio value of the j^{th} simulation. Now, once again if we take $Y^{(j)*} := -Z^{(j)} + (F \cdot S^{(j)})_T$ as we did in (4.2), we may appeal to the result of Buehler *et al.* (2019) explained in (3.6) and rewrite (5.4) as:

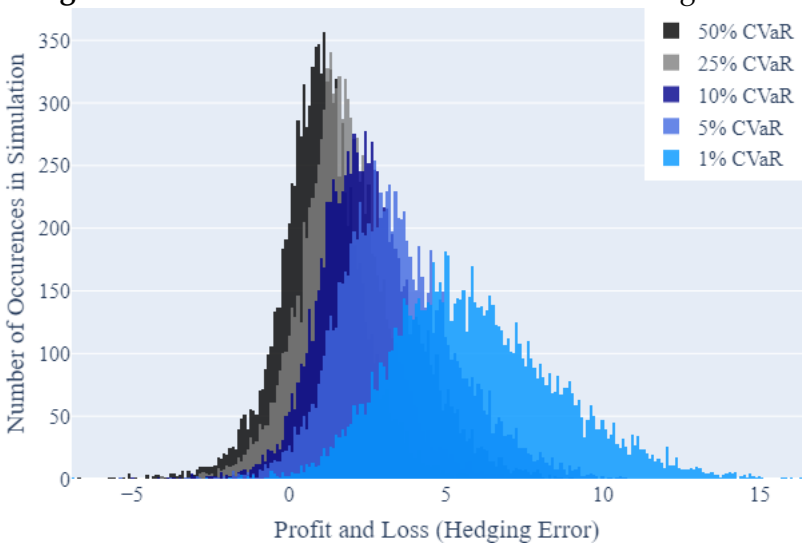
$$\pi(-Z) = \inf_{\theta \in \Theta} \left[-\frac{1}{n_\alpha} \sum_{j=1}^{m} Y^{(j)*} \cdot \mathbb{I}_{\{Y^{(j)*} < Y^\alpha\}} \right]. \tag{5.5}$$

Since we are not squaring the value of Y as in (4.1), the positive value returned by (5.5) is in fact the option price p^* which the model will converge to. However, we may also still follow the approach of (4.3) to arrive at p^* as well.

Now, to showcase the effect which α will have on the pricing process, we can use (4.5) to plot another histogram of our portfolio gains at maturity, but this time for our DHNN which is trained using the CVaR cost function in (5.5). For this illustrative case, we use a sample of 10 000 Monte Carlo paths, with batches of 1 000 trained for 1 000 epochs and $\alpha \in [0.5, 0.25, 0.1, 0.05, 0.01]$.

First, we plot the DHNN portfolio gains for price paths generated according to geometric Brownian motion (GBM) in Figure 5.1[1]. One can immediately notice a more pronounced skew toward positive values of G for decreasing values of α. This makes intuitive sense if we view the parameter α as a proxy for risk-aversion on the part of the market maker: for increased risk aversion (i.e., smaller values of α), a market maker would be relatively more risk-averse toward losses, and would desire the portfolio gains/losses at maturity to be more skewed toward gains ($G > 0$).

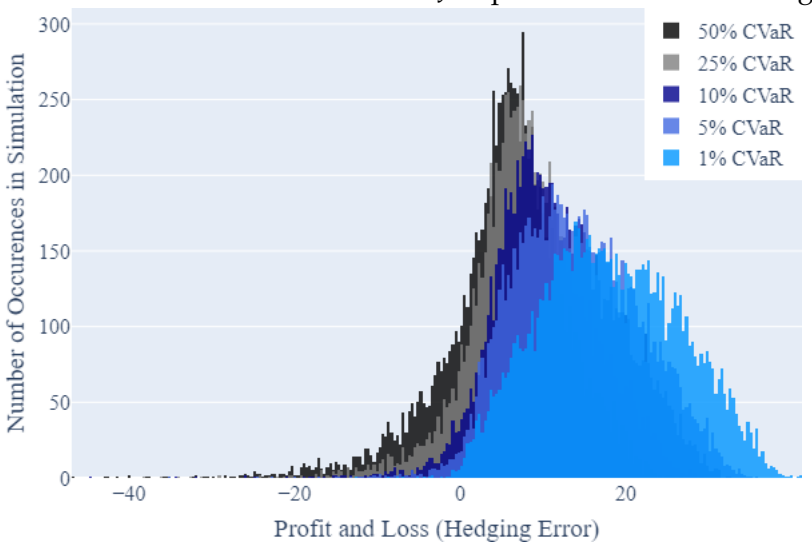
Figure 5.1: PnL Skew Under the GBM Model Using CVaR



[1] We refer to a specific level of CVaR-α as "$\alpha \times 100\%$ CVaR".

Next, we do the same, but for stock prices which evolve according to Merton's jump diffusion model in Figure 5.2. Once again we can observe the skew toward positive G for decreasing values of α. Notice how the left tail of the histograms of G is longer than in the GBM case, as a direct result of the extra source of randomness introduced via the jumps. These jumps will mean larger, less predictable losses, especially if they occur near maturity.

Figure 5.2: PnL Skew Under the Merton Jump-Diffusion Model Using CVaR



5.2 Generalising Break-Even Volatility

Now that we have an idea of how varying α affects δ^* in our DHNN, we would also like to compare how it affects the option price p^* itself, specifically in relation to BEV. But first, we need to change our formulation of BEV to also make use of CVaR (and incorporate α).

Standard BEV minimises the hedging PnL (indeed, it sets it to equal to zero with the choice of volatility) whereas the DHNN tries to do the same in (5.5) but by minimising the average expected loss *past a given threshold* through CVaR. To compare like-for-like we instead rephrase our formulation of BEV in (4.6) as the value of σ which minimises the *CVaR-constrained* hedging profit and loss:

$$\sigma_\alpha^{\text{BE}} = \underset{\sigma\in\mathbb{R}^+}{\arg\min}\, \text{CVaR}_\alpha(\bar{U}(\sigma)), \tag{5.6}$$

with PnL as in (1.4). Similarly to our DHNN approach in (5.5), if we denote U^α as the $(\alpha \times 100)^{\text{th}}$ percentile of profit and loss values, then we can discretise the CVaR

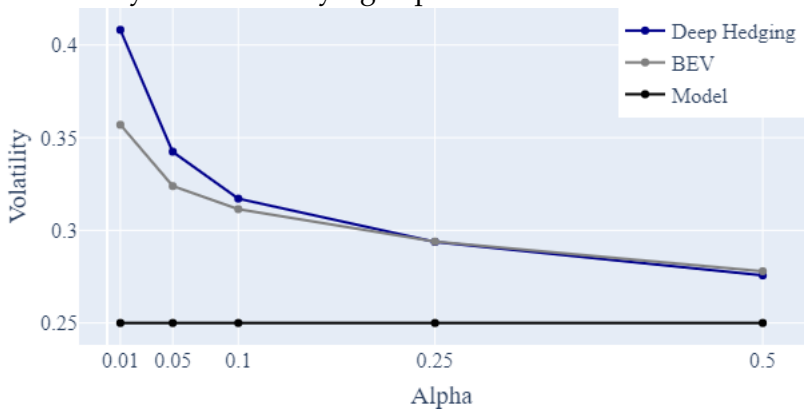
calculation as:

$$\sigma_{\alpha}^{\text{BE}} = \underset{\sigma \in \mathbb{R}^{+}}{\arg\min} \left[-\frac{1}{n_{\alpha}} \sum_{j=1}^{m} U^{(j)}(\sigma) \cdot \mathbb{I}_{\{U^{(j)} < U^{\alpha}\}} \right] \tag{5.7}$$

where $U^{(j)}$ is the profit and loss from the jth price path, n_{α} is the number of profit and loss values less than U^{α} and $\mathbb{I}$ is an indicator function as before.

With the requisite formulations in hand, as a sanity check that both methods are performing as expected, we may now compare BEV against DHNN implied volatility in Figure 5.3 for varying values of α. We are not concerned with the absolute pricing level (i.e., volatility) of the approaches and so use Monte Carlo paths with GBM dynamics as before. Observe that, for both BEV and our DHNN, volatility decreases for increasing values of α. This is exactly what we expect: for decreasing levels of risk aversion, we would expect a market maker to be willing to demand a lower premium for the option.

Figure 5.3: Volatility Skew for Varying Alpha Under the GBM Model Using CVaR



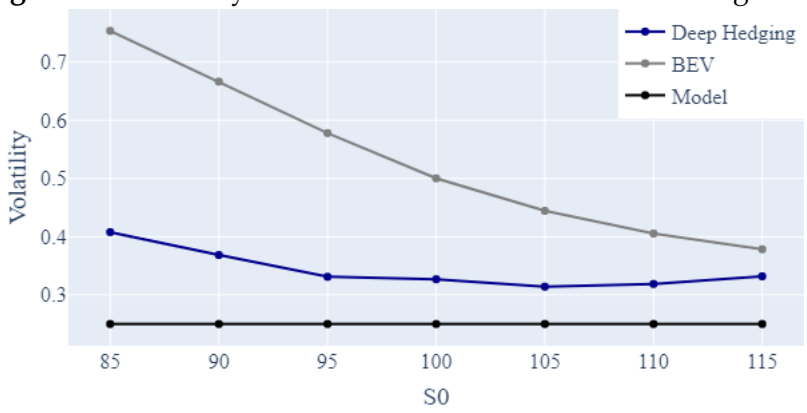
5.3 Risk-Averse Pricing

To conclude our analysis, we compare both BEV and DHNN pricing under a CVaR constraint (with $\alpha = 0.5$) to the model implied volatility (which naturally has no constraint). An α of 0.5 means that we are now essentially targeting the mean PnL below the median PnL amount. Since we are now concerned with realistic pricing levels, we revert back to the overlapping price path simulations used in Chapter 4.

First, for the case of GBM price paths, one can note in Figure 5.4 that the entire implied volatility surface has been shifted up for both the BEV and DHNN approaches. Thus we can immediately observe the risk aversion brought about by using CVaR as our convex risk measure in the form of higher volatility (and thus

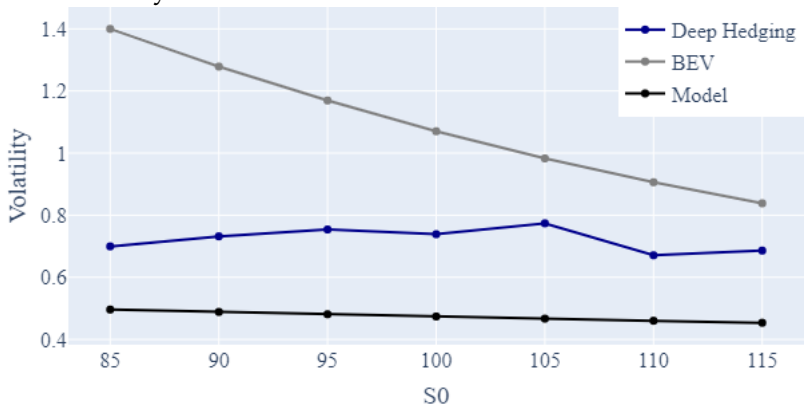
option prices) demanded by the market maker for both BEV and our DHNN. The notable difference between the approaches occurs in the skews produced: BEV produces a far more pronounced skew, charging a much higher premium for further out-the-money values of S_0.

Figure 5.4: Volatility Surfaces Under the GBM Model Using CVaR



The results for Merton jump diffusion price paths in Figure 5.5 are quite similar. Again we note the increased level of risk aversion in general as a result of using CVaR, with both surfaces shifted up markedly in comparison to the risk-neutral model implied surface. We can also observe the same skew for BEV as in the GBM case, where a higher premium is demanded of options which are further out-the-money.

Figure 5.5: Volatility Surfaces Under the Merton Diffusion Model Using CVaR



The skew produced (and higher premium demanded for further out-the-money options) by the BEV approach for both models would seem to imply a 'mispricing' of the risk for us as risk-averse market makers when using a risk-neutral setting, which is now being accounted for in the risk-averse case. However, the fact that our DHNN does not produce the same result (and seems to adjust for this bias in

some way) may make us more skeptical of this claim, and ascribe this observation instead to the formulation of the PnL function in (1.4), which would require disproportionately higher values of σ in order to break even, as there are now fewer paths (with larger losses) in our CVaR generalisation of BEV. In other words, BEV would seem to be less effective when focusing on percentiles of the sample when compared to performing the calculation on the average dynamics of the sample, which could perhaps be due to the rigid model structure of BEV - an area which warrants further research.

Lastly, the absolute levels of volatility implied in Figures 5.4 and 5.5 for both BEV and the DHNN are far too conservative to be able to price competitively in a market setting, and speaks to the need to calibrate α in such a manner that it will strike a balance between the desire of a market maker to price in a prudent enough manner to shield against unexpected losses, but at the same time competititvely enough to still be able to do business with other market participants. However, as was shown in Figure 4.2, using overlapping price paths may skew the results, and lead to difficulties in training and converging to a true optimal solution. Thus the conservative nature of our CVaR volatility surfaces when compared to the model implied approach also comes as a result of using overlapping price paths, and we would expect to see closer surfaces if we had used independent Monte Carlo simulations.

Chapter 6

Conclusion

In our discussion, we have contrasted the existing Break-Even Volatility (BEV) approach toward pricing options formalised by Dupire (2006) to Buehler *et al.* (2019)'s new Deep Hedging (DH) paradigm. The appeal of DH lies largely in its move away from model-dependence (in the sense of BEV being based on Black-Scholes pricing and hedging assumptions) toward a 'model-free' approach which explicitly prices options by finding the optimal hedging strategy to replicate the option liability which the market maker faces at maturity.

First, we examined a risk-neutral setting where we as the market maker are indifferent between hedging profits and losses, and simply seek to minimise our hedging error at maturity. As a justification that our methodology is correct, we have shown that our Deep Hedging Neural Network (DHNN) recovers the Black-Scholes delta hedging parameter in a Black-Scholes world where stock price paths evolve according to geometric Brownian motion (GBM). In this risk-neutral setting, we then compared BEV to our DHNN for time-homogeneous (GBM & Merton jump diffusion) models, and found the performance betwen the two to be comparable and to recover the model-implied results relatively well. We did the same for the time in-homogeneous Heston model, but found the DHNN to perform poorly in this case.

Wanting to acknowledge the residual risk not accounted for in the risk-neutral setting, we next used a convex risk measure, Conditional Value-at-Risk (CVaR), to skew our preferences toward hedging gains (rather than losses) at maturity. The effect of the risk-averse setting which we thus created was illustrated in plotting the histograms of portfolio gains for our DHNN using the CVaR cost function with varying levels of risk-aversion. We also showed how changing risk-aversion through the CVaR-α parameter influences the implied volatility (and thus option price) result, and finally compared BEV and DHNN pricing reults for different moneyness levels and models.

In our analysis, we tried to create realistic data limitations by simulating over-

lapping price paths, which meant that our DHNN portfolio gains were skewed toward longer loss tails as well as both BEV and our DHNN pricing with a more conservative volatiltiy skew. It is here where the use of Generative Adversarial Networks by Wiese *et al.* (2019) to generate Monte Carlo price paths shows promise: one can obtain robust results by training on independent price paths which still have market-realistic dynamics.

In conclusion, given the comparable results between BEV and our DHNN presented above, our simple DHNN (with its model-free advantages) still shows promise against BEV, especially considering (as explained in Chapter 3) that the basic feed-forward architecture may be augmented through miscellaneous recurrence techniques and LSTM architectures.

Bibliography

Abadi, M. and Others (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
URL: *https://www.tensorflow.org/*

Amilon, H. (2003). A neural network versus black-scholes: A comparison of pricing and hedging performances, *Journal of Forecasting* **22**(4): 317–335.

Bakshi, G., Cao, C. and Chen, Z. (1997). Empirical Performance of Alternative Option Pricing Models, *The Journal of Finance* **52**(5): 2003–2049.

Bayer, C., Friz, P. and Gatheral, J. (2016). Pricing under rough volatility, *Quantitative Finance* **16**(6): 887–904.

Black, F. and Scholes, M. (1973). The Pricing of Options and Corporate Liabilities, *Journal of Political Economy* **81**(3): 637–654.

Brannelly, H., Jain, R., Sibanda, P. and Sylvester, M. (2019). Option Pricing and Hedging with Deep Learning, *Financial Mathematics Team Challenge*, University of Cape Town (AIFMRM).

Buehler, H., Gonon, L., Teichmann, J. and Wood, B. (2019). Deep Hedging, *Quantitative Finance* **19**(8): 1271–1291.

Chollet, F. (2015). Keras.
URL: *https://keras.io*

Crawford, D. A. (2019). *Estimating long term equity implied volatility*, Master's thesis, University of Cape Town (AIFMRM).

Devarakonda, A., Naumov, M. and Garland, M. (2017). Adabatch: Adaptive batch sizes for training deep neural networks, *CoRR* **abs/1712.02029**.
URL: *http://arxiv.org/abs/1712.02029*

Dupire, B. (1994). Pricing with a smile, *Risk* **7**(1): 18–20.

Dupire, B. (2006). Skew Modelling, *Bloomberg LP*, New York.

Fliess, M. and Join, C. (2010). Delta hedging in financial engineering: Towards a model-free approach, *18th Mediterranean Conference on Control and Automation, MED'10*, pp. 1429–1434.

Föllmer, H. and Schied, A. (2011). *Stochastic Finance: An Introduction in Discrete Time*, Walter de Gruyter.

Föllmer, H. and Sondermann, D. (1985). Hedging of non-redundant contingent claims.

Fouque, J.-P., Papanicolaou, G. and Sircar, K. R. (2000). Mean-Reverting Stochastic Volatility, *International Journal of Theoretical and Applied Finance* **3**(01): 101–142.

Gatheral, J. (2011). *The Volatility Surface: A Practitioner's Guide*, Vol. 357, John Wiley & Sons.

Gatheral, J. and Jacquier, A. (2011). Convergence of Heston to SVI, *Quantitative Finance* **11**(8): 1129–1132.

Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media.

Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y. (2016). *Deep Learning*, Vol. 1, MIT press Cambridge.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. and Schmidhuber, J. (2016). Lstm: A search space odyssey, *IEEE transactions on neural networks and learning systems* **28**(10): 2222–2232.

Hafner, W. and Zimmermann, H. (2009). *Vinzenz Bronzin's option pricing models: exposition and appraisal*, Springer Science & Business Media.

Hagan, P. S., Kumar, D., Lesniewski, A. S. and Woodward, D. E. (2002). Managing Smile Risk, *The Best of Wilmott* **1**: 249–296.

Halperin, I. (2019). The QLBS Q-Learner Goes NuQLear: Fitted Q Iteration, Inverse RL, and Option Portfolios, *Quantitative Finance* **19**(9): 1543–1553.

Halperin, I. (2020). QLBS: Q-Learner in the Black-Scholes (-Merton) Worlds, *The Journal of Derivatives* .

Hanin, B. (2018). Which neural net architectures give rise to exploding and vanishing gradients?, *Advances in neural information processing systems* **31**.

Heston, S. L. (1993). A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options, *The Review of Financial Studies* **6**(2): 327–343.

Hoffer, E., Hubara, I. and Soudry, D. (2018). Train longer, generalize better: closing the generalization gap in large batch training of neural networks.

Hornik, K., Stinchcombe, M. and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, *Neural networks* **3**(5): 551–560.

Hull, J. C. (2015). *Options, Futures and Other Derivatives*, Vol. 9, Pearson.

Hull, J. and White, A. (2017). Optimal delta hedging for options, *Journal of Banking & Finance* **82**: 180–190.
URL: *https://www.sciencedirect.com/science/article/pii/S0378426617301085*

Hulley, H. and McWalter, T. (2008). Quadratic Hedging of Basis Risk, *Technical report*, Quantitative Finance Research Centre, University of Technology, Sydney.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima, *CoRR* **abs/1609.04836**.
URL: *http://arxiv.org/abs/1609.04836*

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* .

Kisiala, J. (2015). Conditional value-at-risk: Theory and applications, *arXiv preprint arXiv:1511.00140* .

Livieri, G., Mouti, S., Pallavicini, A. and Rosenbaum, M. (2018). Rough volatility: Evidence from option prices, *IISE transactions* **50**(9): 767–776.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks, *arXiv preprint arXiv:1804.07612* .

Matytsin, A. (1999). Modelling Volatility and Volatility Derivatives, *Columbia Practitioners Conference on the Mathematics of Finance*.

Matytsin, A. (2000). Perturbative Analysis of Volatility Smiles, *Work. Pap* .

Medsker, L. R. and Jain, L. (2001). *Recurrent Neural Networks*, CRC Press LLC.

Merton, R. C. (1973). Theory of rational option pricing, *The Bell Journal of economics and management science* pp. 141–183.

Mitoulis, N. (2019). *Break-even volatility*, Master's thesis, University of Cape Town (AIFMRM).

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*, Vol. 25, Determination press San Francisco, CA.

Pflug, G. C. (2000). Some remarks on the value-at-risk and the conditional value-at-risk, *Probabilistic constrained optimization*, Springer, pp. 272–281.

Philipp, G., Song, D. and Carbonell, J. G. (2018). The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions.

Rockafellar, R. T. and Uryasev, S. (2000). Optimization of conditional value-at-risk, *The journal of risk* **2**(3): 21–41.

Rouah, F. D. and Heston, S. L. (2013). *The Heston Model and Its Extensions in Matlab and C#*, Wiley finance, John Wiley & Sons, Incorporated, Somerset.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*, 3rd edn, Prentice Hall Press, USA.

Scarselli, F. and Chung Tsoi, A. (1998). Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results, *Neural networks* **11**(1): 15–37.

Schweizer, M. (1991). Option hedging for semimartingales, *Stochastic Processes and their Applications* **37**(2): 339–363.
URL: *https://www.sciencedirect.com/science/article/pii/030441499190053F*

Smith, S. L., Kindermans, P.-J., Ying, C. and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size, *arXiv preprint arXiv:1711.00489* .

Taleb, N. N. (1997). *Dynamic Hedging: Managing Vanilla and Exotic Options*, Vol. 64, John Wiley & Sons.

Wang, D.-M. (2010). *Monte Carlo Simulations for Complex Option Pricing*, PhD thesis, The University of Manchester (United Kingdom).

Watkins, C. J. and Dayan, P. (1992). Q-Learning, *Machine learning* **8**(3-4): 279–292.

Wiese, M., Bai, L., Wood, B. and Buehler, H. (2019). Deep hedging: learning to simulate equity option markets, *Available at SSRN 3470756* .

Wilmott, P. (1998). *Derivatives: The Theory and Practice of Financial Engineering*, John Wiley & Sons.

Wilmott, P. (2013). *Paul Wilmott on Quantitative Finance*, John Wiley & Sons.

Wilmott, P., Howson, S., Howison, S., Dewynne, J. *et al. (1995).* The Mathematics of Financial Derivatives: A Student Introduction*, Cambridge University Press.*

Yalincak, O. H. (2012). Criticism of the black-scholes model: But why is it still used?:(the answer is simpler than the formula), The Answer is Simpler than the Formula)(July 22, 2012) .

Yamai, Y. and Yoshiba, T. (2005). Value-at-risk versus expected shortfall: A practical perspective, Journal of Banking & Finance **29***(4): 997–1015.*

Appendix A

Stock Price Simulation

A.1 Model Parameters

Description	Variable	Value
Geometric Brownian Motion		
Volatility	σ	0.25
Merton Model		
Volatility without jumps	σ	0.25
Average number of jumps in $[0, T]$	λ	1.5
Average jump size	μ_j	0.75
Std dev. of jumps	σ_j	0.25
Heston Model		
Initial volatility	ν_0	0.05
Long-run average volatility	θ	0.25
The rate at which ν_t reverts to θ	κ	0.15
Volatility of volatility	σ	1
Correlation between $W^{(1)}$ and $W^{(2)}$	ρ	-0.2

Table A.1: Model Parameters Used for Simulation

A.2 Simulation Algorithms

Geometric Brownian Motion

To generate a matrix $\mathbf{S}_{n_{\text{Sims}} \times n_T}$ of stock prices from t_1 to T, we can simulate (2.7) as:

Algorithm 4: Geometric Brownian Motion Discretisation (σ)

Generate $\mathbf{Z}_{n_{\text{Sims}} \times n_T} \sim \text{N}(0, 1)$
for *timesteps* $k = 1 : n_T$ **do**
  $S_{\cdot,k} = S_0 \exp\left(\sum_{i=1}^{k} \left[\left(r - \frac{1}{2}\sigma^2\right) \text{d}t + \sigma\sqrt{\text{d}t} Z_{\cdot,i}\right]\right)$
end

This is easily vectorised in a coding implementation using a `cumsum` command.

Merton Jump Diffusion Model

To generate a matrix $\mathbf{S}_{n_{\text{Sims}} \times n_T}$ of stock prices from t_1 to T, we can simulate (2.8) (see Wang (2010) for a detailed exposition of discretisation techniques) as:

Algorithm 5: Merton Jump Diffusion Model Discretisation $(\sigma, \lambda, j_\mu, j_\sigma)$

Generate $\mathbf{N}_{n_{\text{Sims}} \times n_T} \sim \text{Poi}(\lambda \text{d}t)$
Generate $\mathbf{Z}^{(1)}_{n_{\text{Sims}} \times n_T} \sim \text{N}(0, 1)$
$\mathbf{J} = \mu_j \mathbf{N} + \sigma_j \sqrt{\mathbf{N}} \mathbf{Z}^{(1)}$
Generate $\mathbf{Z}^{(2)}_{n_{\text{Sims}} \times n_T} \sim \text{N}(0, 1)$
for *timesteps* $k = 1 : n_T$ **do**
  $S_{\cdot,k} = S_0 \exp\left(\sum_{i=1}^{k} \left[\left(r - \frac{1}{2}\sigma^2\right) \text{d}t + \sigma \sqrt{\text{d}t} Z^{(2)}_{\cdot,i} + J_{\cdot,i}\right]\right)$
end

Again, this `for` loop is easily vectorised in a coding implementation using a `cumsum` command. One can note here explicitly the Merton jump diffusion model's augmentation of geometric Brownian motion by adding another source of randomness through jumps.

Heston Model

To generate a matrix $\mathbf{S}_{n_{\text{Sims}} \times n_T}$ of stock prices from t_1 to T, we can simulate (2.9) using the Milstein scheme (see, for example, Rouah and Heston (2013)) as:

Algorithm 6: Heston Model Discretisation $(\nu_0, \theta, \kappa, \sigma, \rho)$

Set $\mathbf{\Sigma} = \begin{pmatrix} \rho & 0 \\ 0 & \rho \end{pmatrix}$
Set $\mathbf{L}$ as the Cholesky decomposition of $\mathbf{\Sigma}$
for *simulations* $j = 1 : n_{Sims}$ **do**
  Generate $\mathbf{Z}_{2 \times n_T} \sim \text{N}(0, 1)$
  $\mathbf{X} = \mathbf{L}\mathbf{Z}$
  for *timesteps* $k = 1 : n_T$ **do**
    if $k = 1$ **then**
      $\nu_{j,k-1} = \nu_0$
      $S_{j,k-1} = S_0$
    end
    $\nu_{j,k} =$
    $\left(\nu_{j,k-1} + \kappa(\theta - \nu_{j,k-1})\text{d}t + \sigma \sqrt{\nu_{j,k-1}\text{d}t} X_{1,k} + \frac{1}{4}\sigma^2 (X^2_{1,k} - 1)\text{d}t\right)^+$
    $S_{j,k} = S_{j,k-1} \exp\left[\left(r - \frac{1}{2}\nu_{j,k-1}\right) \text{d}t + \sqrt{\nu_{j,k-1}\text{d}t} X_{2,k}\right]$
  end
end

Appendix B

Deep Hedging Neural Network Hyperparamters

B.1 Architecture

Description	Variable	Value
Layers	L	3
Input dimension	N_0	1
Hidden layer dimension	N_1, N_2	32
Hidden layer activation functions	$\sigma_1, \sigma_2 : \tanh$	$\sigma_l(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Output dimension	N_3	1
Output layer activation function	σ_3 : sigmoid	$\sigma_l(x) = \frac{1}{1+e^{-x}}$

Table B.1: Feed-forward neural network architecture for F_k

B.2 Training

Description	Training Set Size	Batche Size	Epochs	Simulation Type
Figure 4.1	100 000	10 000	1 000	Monte Carlo
Figures 4.2,4.3,4.4,4.5,5.4,5.5	7 438	743	1 000	Overlapping
Figures 5.1, 5.2, 5.3	10 000	1 000	1 000	Monte Carlo

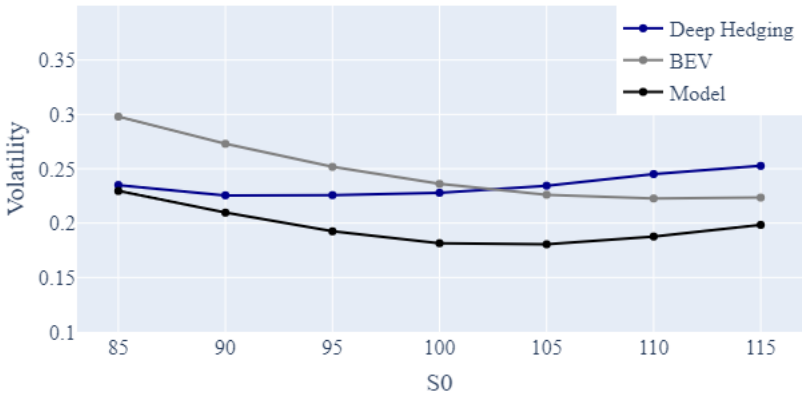
Table B.2: Formulation of Training Sets for the DHNN

Appendix C

Heston Monte Carlo

One can note the improvement in the performance of our Deep Hedging Neural Network (DHNN) when training on Monte Carlo paths. The improvement comes

Figure C.1: Monte Carlo Volatility Comparison Under the Heston Model



about as a result of there being a rich enough data set of independent paths which allow the DHNN to better learn how to hedge the random volatility exhibited by the Heston paths.